DSA

上面所描述的 ElGamal 签名算法在实际中并不常用，更常用的是其变体 DSA。

基本原理

密钥生成

1. 选择一个合适的哈希函数，目前一般选择 SHA1，当前也可以选择强度更高的哈希函数 H。
2. 选择密钥的长度 L 和 N，这两个值决定了签名的安全程度。在最初的 DSS（**Digital Signature Standard** ）中建议 L 必须为 64 的倍数，并且 $512 \leq L \leq 1024$ ，当然，也可以更大。N 必须大小必须不大于哈希函数 H 输出的长度。FIPS 186-3 给出了一些建议的 L 和 N 的取值例子：(1024, 160)，(2048, 224)，(2048, 256)，以及 (3,072, 256)。
3. 选择 N 比特的素数 q。
4. 选择 L 比特的素数 p，使得 p-1 是 q 的倍数。
5. 选择满足$g^k \equiv 1 \bmod p$ 的最小正整数 k 为 q 的 g，即在模 p 的背景下，ord(g)=q 的 g。即 g 在模 p 的意义下，其指数次幂可以生成具有 q 个元素的子群。这里，我们可以通过计算$g = h^{\frac{p-1}{q}} \bmod p$ 来得到 g，其中$1 < h < p - 1$ 。
6. 选择私钥 x，$0 < x < q$ ，计算$y \equiv g^x \bmod p$ 。

公钥为 (p,q,g,y)，私钥为 (x)。

签名

签名步骤如下

1. 选择随机整数数 k 作为临时密钥，$0 < k < q$ 。
2. 计算$r \equiv (g^k \bmod p) \bmod q$
3. 计算$s \equiv (H(m) + xr)k^{-1} \bmod q$

签名结果为 (r,s)。需要注意的是，这里与 Elgamal 很重要的不同是这里使用了哈希函数对消息进行了哈希处理。

验证

验证过程如下

1. 计算辅助值，$w = s^{-1} \bmod q$
2. 计算辅助值，$u_1 = H(m)w \bmod q$
3. 计算辅助值，$u_2 = rw \bmod q$
4. 计算$v = (g^{u_1}y^{u_2} \bmod p) \bmod q$
5. 如果 v 与 r 相等，则校验成功。

正确性推导

首先，g 满足 $g^k \equiv 1 \bmod p$ 的最小正整数 k 为 q。所以 $g^q \equiv 1 \bmod p$ 。所以 $g^x \equiv g^{x \bmod q} \bmod p$ 。进而

$v = (g^{u_1}y^{u_2} \bmod p) \bmod q = g^{u_1}g^{xu_2} \equiv g^{H(m)w}g^{xrw} \equiv g^{H(m)w+xrw}$

又$s \equiv (H(m) + xr)k^{-1} \bmod q$ 且$w = s^{-1} \bmod q$ 所以

$k \equiv s^{-1}(H(m) + xr) \equiv H(m)w + xrw \bmod q$

所以$v \equiv g^k$ 。正确性得证。

安全性

已知 k

原理

如果知道了随机密钥 k，那么我们就可以根据$s \equiv (H(m) + xr)k^{-1} \bmod q$ 计算私钥 d，几乎攻破了 DSA。

这里一般情况下，消息的 hash 值都会给出。

$x \equiv r^{-1}(ks - H(m)) \bmod q$

k 共享

原理

如果在两次签名的过程中共享了 k，我们就可以进行攻击。

假设签名的消息为 m1,m2，显然，两者的 r 的值一样，此外

$s_1 \equiv (H(m_1) + xr)k^{-1} \bmod q$

$s_2 \equiv (H(m_2) + xr)k^{-1} \bmod q$

这里我们除了 x 和 k 不知道剩下的均知道，那么

$s_1k \equiv H(m_1) + xr$

$s_2k \equiv H(m_2) + xr$

两式相减

$k(s_1 - s_2) \equiv H(m_1) - H(m_2) \bmod q$

此时 即可解出 k，进一步我们可以解出 x。

例子

这里我们以湖湘杯的 DSA 为例，但是不能直接去做，，，因为发现在验证 message4 的时候签名不通过。源题目我没有了，。 ，，这里我以 Jarvis OJ 中经过修改的题目 DSA 为例

```
➜  2016湖湘杯DSA git:(master) ✗ openssl sha1 -verify dsa_public.pem -signature packet1/sign1.bin  packet1/message1
Verified OK
➜  2016湖湘杯DSA git:(master) ✗ openssl sha1 -verify dsa_public.pem -signature packet2/sign2.bin  packet2/message1
packet2/message1: No such file or directory
➜  2016湖湘杯DSA git:(master) ✗ openssl sha1 -verify dsa_public.pem -signature packet2/sign2.bin  packet2/message2
Verified OK
➜  2016湖湘杯DSA git:(master) ✗ openssl sha1 -verify dsa_public.pem -signature packet3/sign3.bin  packet3/message3
Verified OK
➜  2016湖湘杯DSA git:(master) ✗ openssl sha1 -verify dsa_public.pem -signature packet4/sign4.bin  packet4/message4
Verified OK
```

可以看出四则消息全部校验通过。这里之所以会联想到共享 k 是因为题目中提示了 PS3 的破解曾用到这个方法，从网上搜索可知该攻击。

下面，我们看一下签名后的值，这里使用的命令如下

```
➜  2016湖湘杯DSA git:(master) ✗ openssl asn1parse -inform der -in packet4/sign4.bin
    0:d=0  hl=2 l=  44 cons: SEQUENCE
    2:d=1  hl=2 l=  20 prim: INTEGER           :5090DA81FEDE048D706D80E0AC47701E5A9EF1CC
   24:d=1  hl=2 l=  20 prim: INTEGER           :5E10DED084203CCBCEC3356A2CA02FF318FD4123
➜  2016湖湘杯DSA git:(master) ✗ openssl asn1parse -inform der -in packet3/sign3.bin
    0:d=0  hl=2 l=  44 cons: SEQUENCE
    2:d=1  hl=2 l=  20 prim: INTEGER           :5090DA81FEDE048D706D80E0AC47701E5A9EF1CC
   24:d=1  hl=2 l=  20 prim: INTEGER           :30EB88E6A4BFB1B16728A974210AE4E41B42677D
➜  2016湖湘杯DSA git:(master) ✗ openssl asn1parse -inform der -in packet2/sign2.bin
    0:d=0  hl=2 l=  44 cons: SEQUENCE
    2:d=1  hl=2 l=  20 prim: INTEGER           :60B9F2A5BA689B802942D667ED5D1EED066C5A7F
   24:d=1  hl=2 l=  20 prim: INTEGER           :3DC8921BA26B514F4D991A85482750E0225A15B5
➜  2016湖湘杯DSA git:(master) ✗ openssl asn1parse -inform der -in packet1/sign1.bin
    0:d=0  hl=2 l=  45 cons: SEQUENCE
    2:d=1  hl=2 l=  21 prim: INTEGER           :8158B477C5AA033D650596E93653C730D26BA409
   25:d=1  hl=2 l=  20 prim: INTEGER           :165B9DD1C93230C31111E5A4E6EB5181F990F702
```

其中，获取的第一个值是 r，第二个值是 s。可以看到第 4 个 packet 和第 3 个 packet 共享了 k，因为他们的 r 一致。

这里我们可以使用 openssl 看下公钥

```
➜  2016湖湘杯DSA git:(master) ✗ openssl dsa -in dsa_public.pem -text -noout  -pubin
read DSA key
pub:
```

```
    45:bb:18:f6:0e:b0:51:f9:d4:82:18:df:8c:d9:56:
    33:0a:4f:f3:0a:f5:34:4f:6c:95:40:06:1d:53:83:
    29:2d:95:c4:df:c8:ac:26:ca:45:2e:17:0d:c7:9b:
    e1:5c:c6:15:9e:03:7b:cc:f5:64:ef:36:1c:18:c9:
    9e:8a:eb:0b:c1:ac:f9:c0:c3:5d:62:0d:60:bb:73:
    11:f1:cf:08:cf:bc:34:cc:aa:79:ef:1d:ad:8a:7a:
    6f:ac:ce:86:65:90:06:d4:fa:f0:57:71:68:57:ec:
    7c:a6:04:ad:e2:c3:d7:31:d6:d0:2f:93:31:98:d3:
    90:c3:ef:c3:f3:ff:04:6f
P:
    00:c0:59:6c:3b:5e:93:3d:33:78:be:36:26:be:31:
    5e:e7:0c:a6:b5:b1:1a:51:9b:55:23:d4:0e:5b:a7:
    45:66:e2:2c:c8:8b:fe:c5:6a:ad:66:91:8b:9b:30:
    ad:28:13:88:f0:bb:c6:b8:02:6b:7c:80:26:e9:11:
    84:be:e0:c8:ad:10:cc:f2:96:be:cf:e5:05:05:38:
    3c:b4:a9:54:b3:7c:b5:88:67:2f:7c:09:57:b6:fd:
    f2:fa:05:38:fd:ad:83:93:4a:45:e4:f9:9d:38:de:
    57:c0:8a:24:d0:0d:1c:c5:d5:fb:db:73:29:1c:d1:
    0c:e7:57:68:90:b6:ba:08:9b
Q:
    00:86:8f:78:b8:c8:50:0b:eb:f6:7a:58:e3:3c:1f:
    53:9d:35:70:d1:bd
G:
    4c:d5:e6:b6:6a:6e:b7:e9:27:94:e3:61:1f:41:53:
    cb:11:af:5a:08:d9:d4:f8:a3:f2:50:03:72:91:ba:
    5f:ff:3c:29:a8:c3:7b:c4:ee:5f:98:ec:17:f4:18:
    bc:71:61:01:6c:94:c8:49:02:e4:00:3a:79:87:f0:
    d8:cf:6a:61:c1:3a:fd:56:73:ca:a5:fb:41:15:08:
    cd:b3:50:1b:df:f7:3e:74:79:25:f7:65:86:f4:07:
    9f:ea:12:09:8b:34:50:84:4a:2a:9e:5d:0a:99:bd:
    86:5e:05:70:d5:19:7d:f4:a1:c9:b8:01:8f:b9:9c:
    dc:e9:15:7b:98:50:01:79
```

下面，我们直接利用上面的原理编写程序即可，程序如下

```python
#coding=utf8
from Crypto.PublicKey import DSA
from hashlib import sha1
import gmpy2
with open('./dsa_public.pem') as f:
    key = DSA.importKey(f)
    y = key.y
    g = key.g
    p = key.p
    q = key.q
f3 = open(r"packet3/message3", 'r')
f4 = open(r"packet4/message4", 'r')
data3 = f3.read()
data4 = f4.read()
sha = sha1()
sha.update(data3)
m3 = int(sha.hexdigest(), 16)
sha = sha1()
sha.update(data4)
m4 = int(sha.hexdigest(), 16)
print m3, m4
s3 = 0x30EB88E6A4BFB1B16728A974210AE4E41B42677D
s4 = 0x5E10DED084203CCBCEC3356A2CA02FF318FD4123
r = 0x5090DA81FEDE048D706D80E0AC47701E5A9EF1CC
ds = s4 - s3
dm = m4 - m3
k = gmpy2.mul(dm, gmpy2.invert(ds, q))
k = gmpy2.f_mod(k, q)
tmp = gmpy2.mul(k, s3) - m3
x = tmp * gmpy2.invert(r, q)
x = gmpy2.f_mod(x, q)
print int(x)
```

我发现 pip 安装的 pycrypto 竟然没有 DSA 的 importKey 函数。。。只好从 github 上下载安装了 pycrypto。。。

结果如下

```
➜  2016湖湘杯DSA git:(master) X python exp.py
110488417796252422117450972681125617714623596155094373513204453614900071076054577862818196184023052079358815380532078342252161514868778508607074 4
```
